Networks
Project 2a
Network Layer Routing

Continuing with the code base from Project-1c, we add a `NetworkLayer`. Doing so requires some restructuring of the `DataLinkLayer`, and of each `Host` and its network stack as a whole, but most of the structure should be familiar. With the `NetworkLayer` in place, we make *routing* possible through multiple links.

1 The simulator

Getting it: To get started, download the simulator's source code:

`https://bit.ly/cosc-283-23f-p2a`

New structure: As with Project-1c, each host is represented by a `Host` object, which then communicates with a `NetworkLayer`. The `NetworkLayer`, in turn, has multiple `DataLinkLayer` objects—one for each link to another host. (Of course, each `DataLinkLayer` is connected to its own `PhysicalLayer`, and pairs of these /PhysicalLayer| objects share a `Medium`.)

Each `Host` is run as a *thread*. A message is read from a file (as usual) and passed to the *sending host*. That host passes the message down to its network layer, which is then responsible for creating packets to contain the data, and sending each packet along one of its links to move it towards the *destination host*. Each host along the way should *forward* the packet. When the packet arrives at its destination, its contents should be delivered to the client (the `Host`).

Running the simulator: Once you have completed the implementation of the `RandomNetworkLayer` (see below, Section 2), begin by seeing the usage of the simulator:

```
$ java Simulator
Usage: java Simulator <medium type>
                      <data link layer type>
                      <network layer type>
                      <links file>
                      <source host>
                      <destination host>
                      <transmission data file>
```

As usual, you need to specify a *medium type* (which should be `Perfect`) and a *data link layer type* (`Dumb`). This combination will avoid the added complexity of error detection and flow control when trying to debug the new network layer code. In specifying a *network layer type*, you will be implementing a `RandomNetworkLayer`, and so you should specify `Random`

here.

The *links file* contains a listing of the direct connections between pairs of hosts. The file takes the following format:

```
Earth Mars 20
Earth Venus 15
Venus Mercury 7
Mars Jupiter 12
Earth Jupiter 55
```

Each row of this file represents a single link between two hosts: the first two fields are strings that provide *hostnames*, and the third field is an integer *connection weight*, representing the cost (in an abstract sense) of communicating over that link.

From this list of links, `Simulator` creates the objects needed: for each new hostname, it creates a `Host` and `NetworkLayer`; for each link, it creates a `Medium`, which it then connects to `PhysicalLayer` and `DataLinkLayer` objects that attach to the given hosts' `NetworkLayer`s. Thus, from the list of links, the entire simulated network is created.

The *source host* and *distination host* are simply hostnames from which, and to which, a message should be sent through the simulated network. The *transmission data file* contains the bytes to be sent. Thus, invoking the simulator should look something like:

```
$ java Simulator Perfect Dumb Random links.txt Venus Jupiter message.txt
```

2 Implementing random routing

The `RandomNetworkLayer` should be completed to implement *randomized routing*. That is, when a packet arrives, a `RandomNetworkLayer` object should determine how to handle it. If the packet is destined for this host, then deliver the data; if the packet is destined for another host, it should **randomly select a link** through which to forward the packet.

The following methods, which are *abstract* in the parent class `NetworkLayer`, need to be completed in `RandomNetworkLayer`:

- `createPacket()`: Given a sequence of bytes and a destination, create a packet from this host (which is the source) to the destination host, returning the bytes of the packet. This is where a *packet header* must be created, and thus a packet format determined.

- `route()`: Given a destination, determine which outgoing link to use in order to send a packet towards that destination.

- `extractPacket()`: Given a queue of received bytes, examine that queue to determine whether a full packet has arrived. If so, extract those bytes from the queue and into an array, returning the array of the complete packet.
- `processPacket()`: Given an array that contains a complete packet, examine its header. If the packet is destined for this host, extract the data from the packet and deliver it to the client `Host`; otherwise, `route()` the packet and re-send it.

3 How to submit your work

Go to GradeScope for our course, where you can submit your work. Notice that **you should only submit** `RandomNetworkLayer.java`. *Don't submit the other source code or class files.*

This assignment is due on Friday, Nov-10, 11:59 pm.